**Workshop: Desarrollo de una Aplicación con NestJS**

**Título: Construyendo una Aplicación Básica de E-commerce con NestJS**

**Objetivo:**

El objetivo de este workshop es guiar a los coders a través de la creación de una pequeña aplicación de e-commerce usando NestJS. A lo largo del workshop, los participantes implementarán autenticación de usuarios usando JWT, manejarán la seguridad con hashing de contraseñas, aplicarán filtros de errores y pipes, y gestionarán la permisología para diferentes roles de usuario (usuario y administrador).

**Agenda:**

1. **Introducción a NestJS y Configuración del Proyecto**
   * Breve introducción a NestJS y su arquitectura.
   * Configuración inicial del proyecto con Nest CLI.
   * Creación de módulos, controladores y servicios básicos.
2. **Modelado de Datos y Configuración de TypeORM**
   * Configuración de TypeORM en NestJS.
   * Creación de entidades User, Product, y Order.
   * Relacionamiento de tablas (User con Order, Order con Product).
3. **Autenticación de Usuarios con JWT**
   * Implementación de la funcionalidad de registro de usuarios con hashing de contraseñas (bcrypt).
   * Creación del módulo de autenticación usando JWT.
   * Generación de tokens JWT para autenticación.
   * Implementación de middleware y guards para proteger rutas con JWT.
4. **Roles y Permisología**
   * Definición de roles de usuario (user, admin).
   * Implementación de guards para controlar acceso a rutas según el rol.
   * Ejemplo: Rutas específicas solo accesibles para admin.
5. **Gestión de Productos y Pedidos**
   * Implementación de CRUD de productos (solo accesible para admin, para user se permite solo leer).
   * Creación y gestión de pedidos por parte de los usuarios.
   * Relación entre usuarios y pedidos.
6. **Manejo de Errores y Validaciones**
   * Creación y uso de Exception Filters para manejo de errores personalizados.
   * Implementación de Pipes para validaciones y transformación de datos.
   * Ejemplo práctico de validaciones con DTOs y class-validator.
7. **Conclusión y Q&A**
   * Resumen de los conceptos aprendidos.
   * Preguntas y respuestas.

**Ejercicio Práctico:**

**Descripción:**  
Desarrollar una pequeña aplicación de e-commerce que permita a los usuarios registrarse, iniciar sesión, agregar productos al catálogo (solo admin), y realizar pedidos. La aplicación debe implementar seguridad usando JWT, aplicar validaciones y errores personalizados con filtros, y gestionar permisología con roles de usuario.

**Requisitos Específicos:**

**Entidad User:**

* **Atributos**: id, email, password, role.
* **Funcionalidades**: Registro, login, recuperación de datos de usuario.
* **Relación**:
  + **Uno a muchos con Order**: Un usuario puede tener múltiples pedidos (Orders).
* **Autenticación** usando JWT, con contraseñas hasheadas.

**Entidad Product:**

* **Atributos**: id, name, price, description.
* **Funcionalidades**: CRUD completo.
* **Relación**:
  + **Muchos a muchos con Order**: Un producto puede estar en múltiples pedidos, y un pedido puede contener múltiples productos.

**Entidad Order:**

* **Atributos**: id, userId, products, totalPrice.
* **Funcionalidades**: Crear pedido (accesible para cualquier usuario autenticado).
* **Relaciones**:
  + **Muchos a uno con User**: Un pedido pertenece a un usuario.
  + **Muchos a muchos con Product**: Un pedido puede contener múltiples productos.

**Permisología y Roles:**

* **Dos roles**: user y admin.
* **Control de acceso basado en roles** usando guards.
* **Rutas protegidas** por JWT y roles.

**Filtros de Errores y Pipes:**

* Implementar un **filtro global** para manejar errores.
* Usar **pipes** para validar datos en los DTOs.

**Descripción de la Estructura:**

1. **auth/**: Este directorio maneja todo lo relacionado con la autenticación.
   * **dto/**: Contiene los Data Transfer Objects (DTO) utilizados para la autenticación, como los DTOs de login y register.
   * **guards/**: Contiene los guards relacionados con la autenticación, como el guard de JWT.
   * **strategies/**: Contiene las estrategias de autenticación, como la estrategia de JWT.
   * **auth.controller.ts**: El controlador que maneja las rutas relacionadas con la autenticación.
   * **auth.module.ts**: El módulo que agrupa todos los componentes de autenticación.
   * **auth.service.ts**: El servicio que contiene la lógica de negocio para la autenticación.
2. **common/**: Este directorio es para funcionalidades que se reutilizan en toda la aplicación.
   * **decorators/**: Contiene decoradores personalizados, como el decorador de roles.
   * **filters/**: Contiene filtros personalizados, como el filtro de excepciones HTTP.
   * **pipes/**: Contiene pipes personalizados, como el pipe de validación.
   * **common.module.ts**: El módulo que agrupa todos los componentes comunes.
3. **users/**: Este directorio maneja todo lo relacionado con los usuarios.
   * **dto/**: Contiene los DTOs utilizados para la creación y actualización de usuarios.
   * **entities/**: Contiene la entidad User, que define el modelo de usuario.
   * **users.controller.ts**: El controlador que maneja las rutas relacionadas con los usuarios.
   * **users.module.ts**: El módulo que agrupa todos los componentes relacionados con usuarios.
   * **users.service.ts**: El servicio que contiene la lógica de negocio relacionada con los usuarios.
4. **products/**: Este directorio maneja todo lo relacionado con los productos.
   * **dto/**: Contiene los DTOs utilizados para la creación y actualización de productos.
   * **entities/**: Contiene la entidad Product, que define el modelo de producto.
   * **products.controller.ts**: El controlador que maneja las rutas relacionadas con los productos.
   * **products.module.ts**: El módulo que agrupa todos los componentes relacionados con productos.
   * **products.service.ts**: El servicio que contiene la lógica de negocio relacionada con los productos.
5. **orders/**: Este directorio maneja todo lo relacionado con los pedidos.
   * **dto/**: Contiene el DTO utilizado para la creación de pedidos.
   * **entities/**: Contiene la entidad Order, que define el modelo de pedido.
   * **orders.controller.ts**: El controlador que maneja las rutas relacionadas con los pedidos.
   * **orders.module.ts**: El módulo que agrupa todos los componentes relacionados con pedidos.
   * **orders.service.ts**: El servicio que contiene la lógica de negocio relacionada con los pedidos.
6. **app.\***: Estos archivos son la entrada principal de la aplicación.
   * **app.controller.ts**: Controlador principal de la aplicación.
   * **app.module.ts**: Módulo principal que importa y organiza todos los demás módulos.
   * **app.service.ts**: Servicio principal de la aplicación.
   * **main.ts**: Punto de entrada de la aplicación donde se arranca el servidor.

